Training DL

Deep Learning (DSE316/616)

Vinod K Kurmi
Assistant Professor, DSE

Indian Institute of Science Education and Research Bhopal

Aug 29, 2022

Disclaimer

- Much of the material and slides for this lecture were borrowed from
  - Bernhard Schölkopf's MLSS 2017 lecture,
  - Tommi Jaakkola's 6.867 class,
  - CMP784: Deep Learning Fall 2021 Erkut Erdem Hacettepe University
  - Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class
  - Hongsheng Li's ELEG5491 class
  - Tsz-Chiu Au slides
  - Mitesh Khapra Class notes

Previous class: Momentum based Gradient Descent

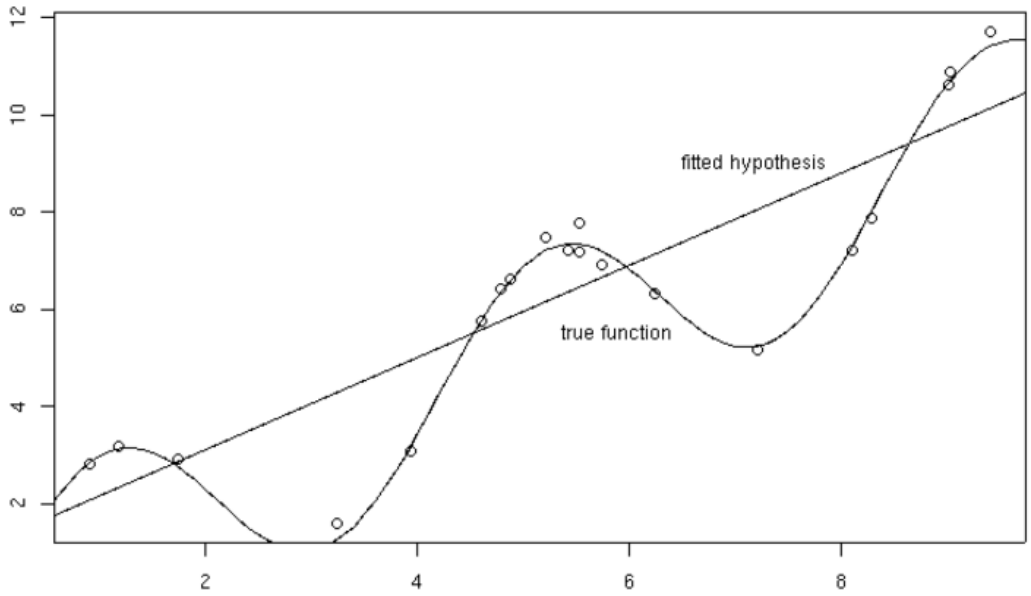
Some observations about gradient descent

- It takes a lot of time to navigate regions having a gentle slope
- This is because the gradient in these regions is very small Can we do something better ?
- Yes, let's take a look at **'Momentum based gradient descent'**

Review: Bias and Variance

Suppose that we generate 20 samples as follows, and use linear regression to fit the generated data.

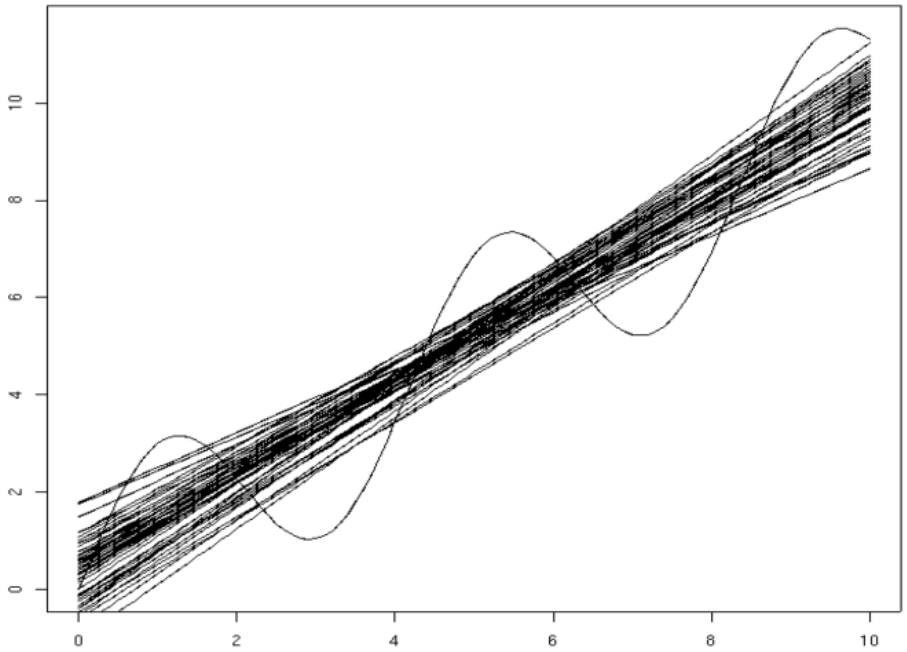
$$y = x + 2\sin(1.5x) + N(0,0.2)$$



Review: Bias and Variance

Example

If we perform this 50 times, then the line fitting to generated samples will be different at each run.



Generalization Error

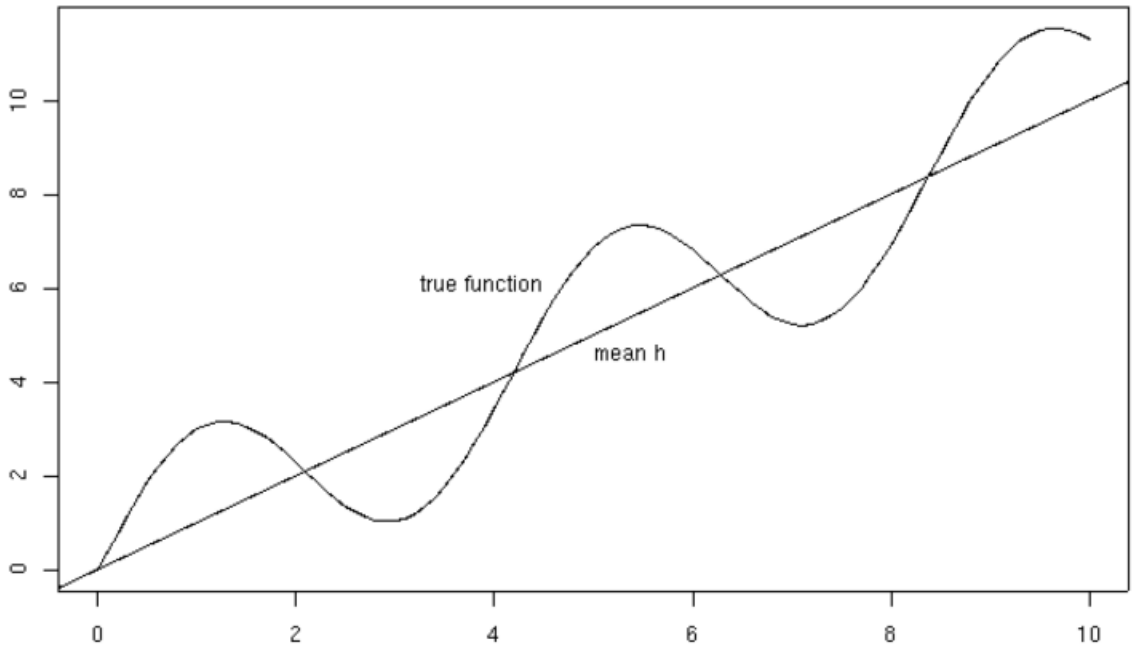
Given a new data point x, what is the expected prediction error of the hypothesis h?

$$E\left[\left(y - h(x)\right)^2\right]$$

We will learn to derive this decomposition of the expected error into bias, variance, and noise terms.

Bias

Discrepancy between averaged estimated and true function



$$Bias[h(x)] = \overline{h(x)} - f(x)$$

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

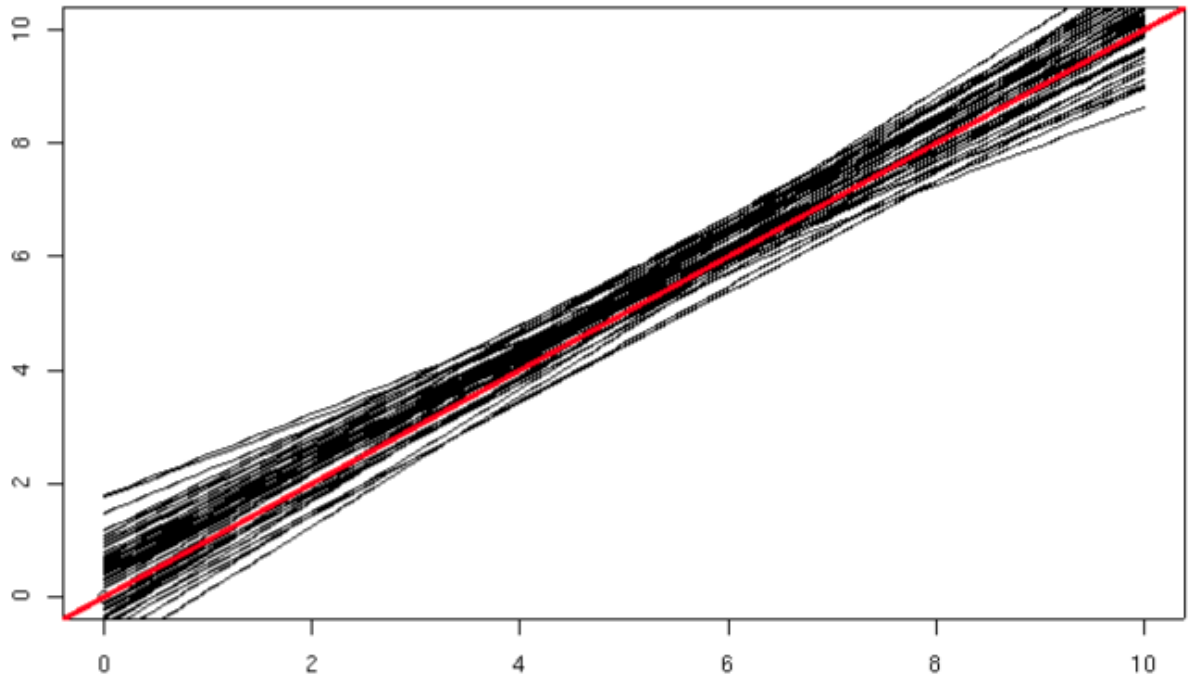
- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Source of Bias

- Inability to represent certain decision boundaries
  - e.g.) linear classifier, decision trees.
- Incorrect assumptions
  - e.g.) linear assumption for data generated from a higherdegree polynomial model.
- Models that are too global (or too smooth)
  - e.g.) a single linear separator
- If the bias is high, the model is underfitting the data.

Variance

Discrepancy between models trained on different training sets



$$Var[h(x)] = E_P\left[\left(h(x) - \overline{h(x)}\right)^2\right]$$

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

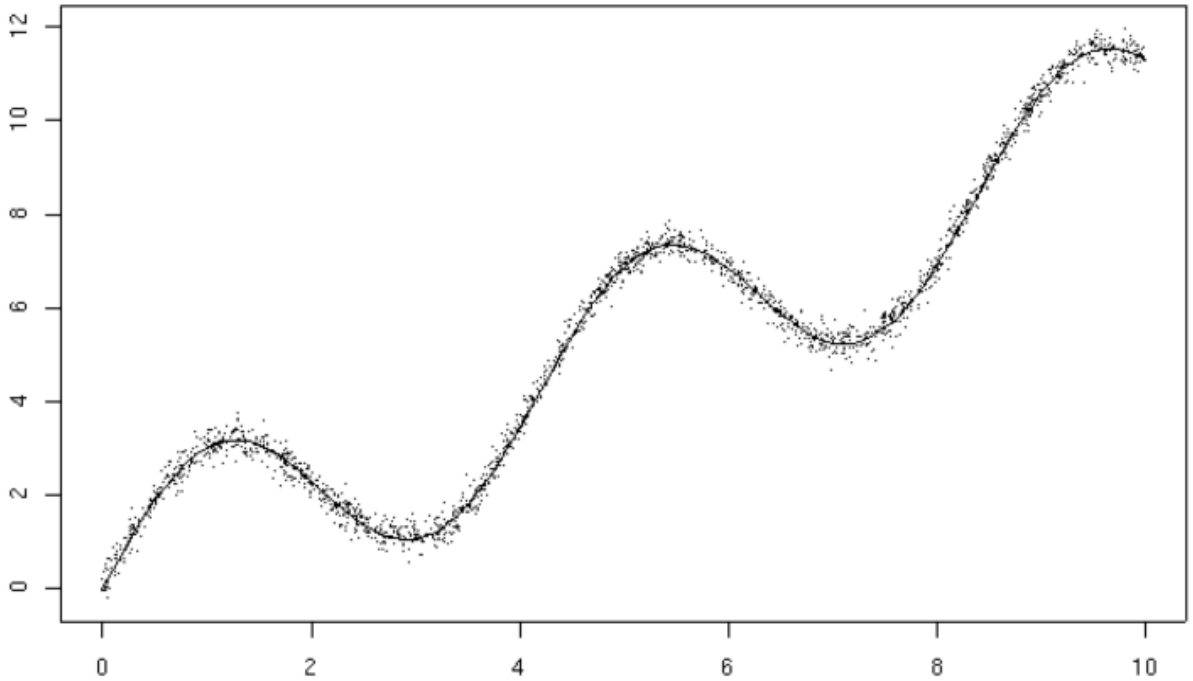
- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Source of Variance

- Statistical sources
  - Classifiers that are too local and can easily fit the data.
  - e.g.) nearest neighbor, large decision trees.
- Computational sources
  - Learning algorithms that make sharp decisions can be unstable, as the decision boundary can change if one training example changes.
  - Randomization in the learning algorithm
  - e.g.) neural networks with random initial weights.
- If the variance is high, the model is overfitting the data.

Noise

Irreducible error that describes how y varies from $f(x)$



$$Noise[h(x)] = E\left[\left(y - f(x)\right)^2\right] = E[\epsilon^2] = \sigma^2$$

Bias / Variance and Model Complexity

$$E_P\left[\left(y - h(x)\right)^2\right]$$

$$= E_P[h(x)^2 - 2yh(x) + y^2]$$

$$= E_P[h(x)^2] + E_P[y^2] - 2E_P[y]E_P[h(x)]$$

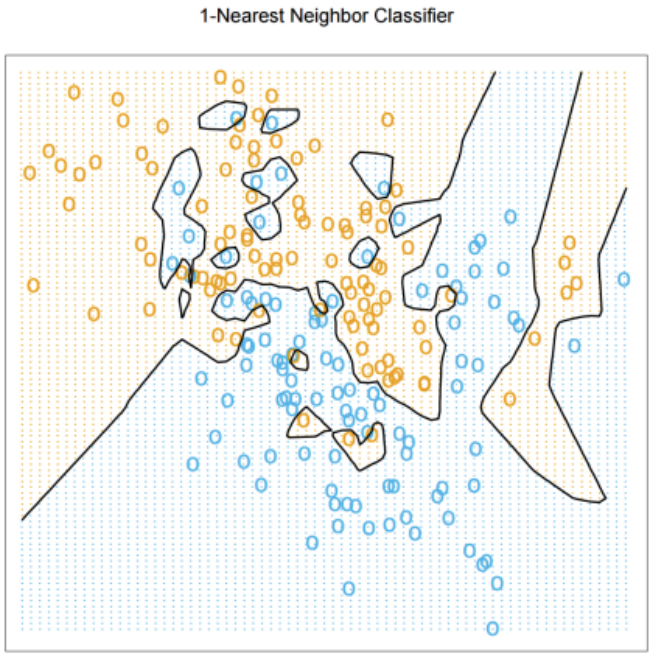
$$= E_P\left[\left(h(x) - \overline{h(x)}\right)^2\right] + \overline{h(x)}^2 + E_P\left[\left(y - f(x)\right)^2\right] + f(x)^2 - 2f(x)\overline{h(x)}$$

$$= E_P\left[\left(h(x) - \overline{h(x)}\right)^2\right] + \left(\overline{h(x)} - f(x)\right)^2 + E_P\left[\left(y - f(x)\right)^2\right]$$

Variance | Bias^2 | Noise

Bias-Variance Decomposition

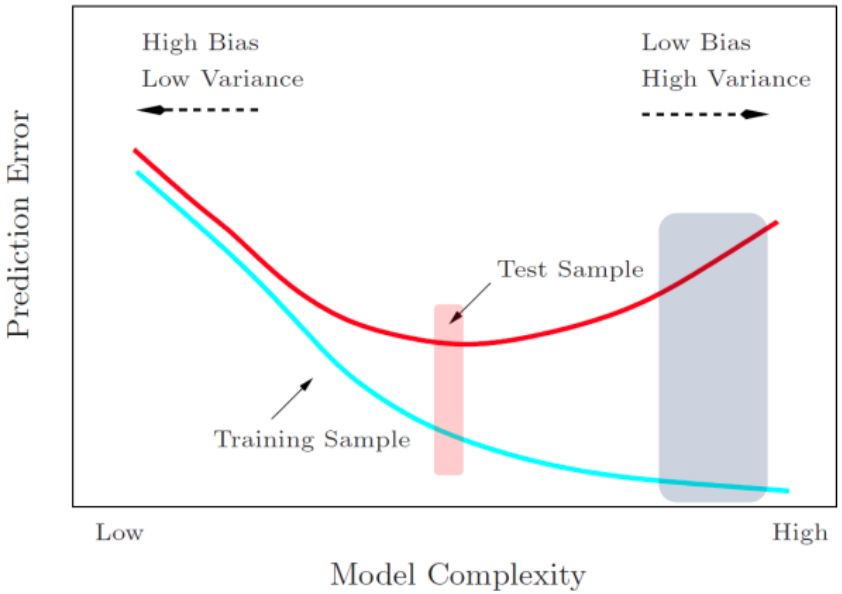
Simple models have high bias, but low variance.

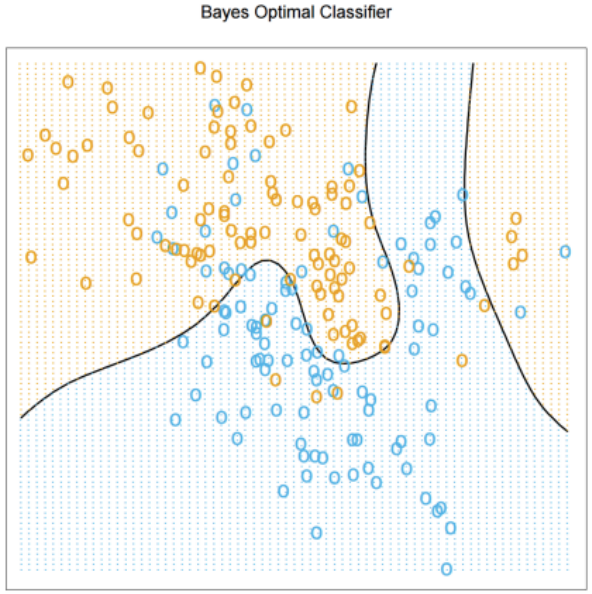


Complex models have low bias, but high variance.

Bias / Variance Trade-off

We want to reduce both sources of error to obtain a model that is 'just right'.





However, finding the right complexity of the model is not a simple matter.

Most regularization methods aim to reduce variance at the cost of added bias.

Norm-based Regularizations

Regularization in General Machine Learning Context

Introduction of penalty terms to guide the learning with some additional information.

$$\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \lambda\Omega(\boldsymbol{w})$$

Regularization term

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{w}, x_i, y_i)$$

One type of penalty term we can add in, is a norm of the parameter $\boldsymbol{w}$, which can bring in many nice effects into the learning process.

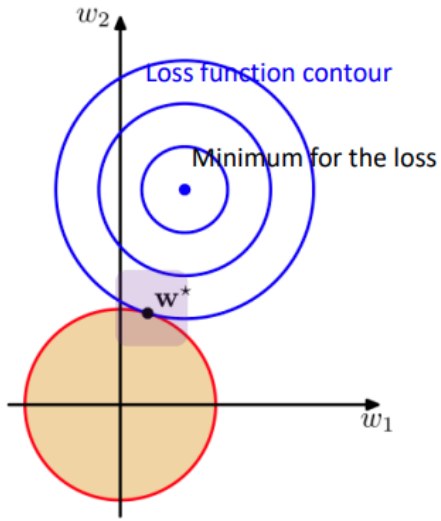
L2 Regularization

L2 regularizations, based on 2 norm, is one the most common types of regularizations

$$J(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

$$\|\boldsymbol{w}\|_2 = \sqrt{\sum_{w_j} |w_j|^2}$$

$$\|\boldsymbol{w}\|_2 \leq \lambda$$



Shrinks the value of the variables while favoring similar weights among them

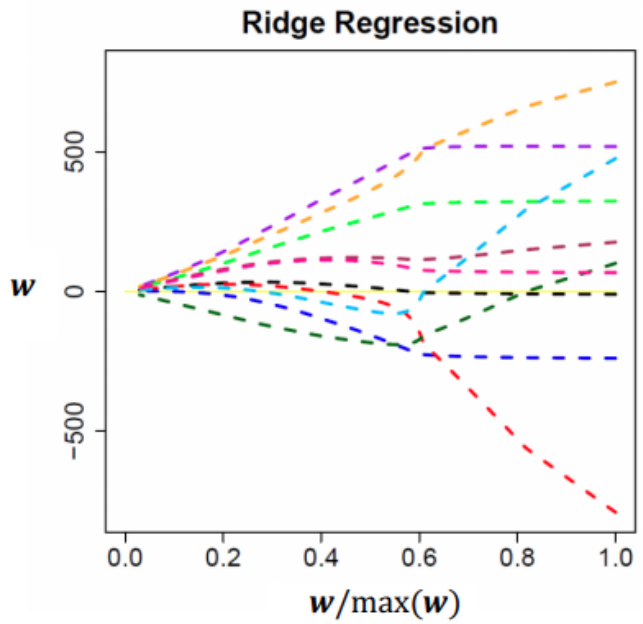
Ridge Regression

Linear regression with L2-regularization

$$\min_w \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\Omega(\boldsymbol{w})$$

$$\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T y$$



Shrinks all variables to zero – reduces variance while introducing bias.

L2 Regularization

Often referred to as ***weight decay*** regularizer, as optimizing for l2-norm will shrink the value of the variables at each iteration.

Regularized objective $\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}$

Corresponding gradient $\nabla_{\mathrm{w}}\tilde{J}(\boldsymbol{w}) = \nabla_{\mathrm{w}}J(\boldsymbol{w}) + \alpha\boldsymbol{w}$

Update at each gradient step

$$w \leftarrow w - \epsilon\big(\lambda w + \nabla_{\mathrm{w}}J(\boldsymbol{w})\big)$$
$$w \leftarrow (1 - \epsilon\lambda)w - \epsilon\nabla_{\mathrm{w}}J(\boldsymbol{w})$$

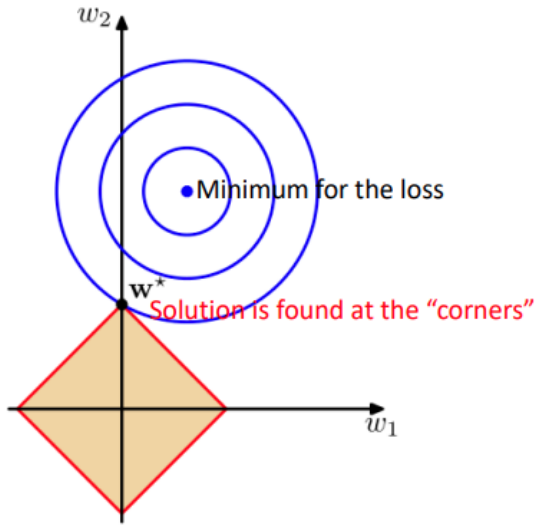
At each step, the weight decay term will shrink the weight by a constant factor.

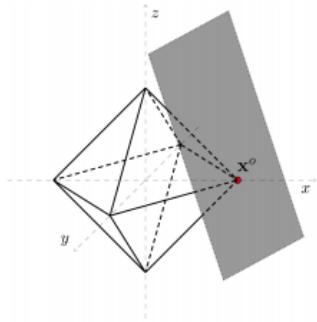
L1 regularization

$$\min_{w} J(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

$$\|\boldsymbol{w}\|_1 = \sum_{w_j} |w_j|$$

$$\|\boldsymbol{w}\|_1 \leq \lambda$$





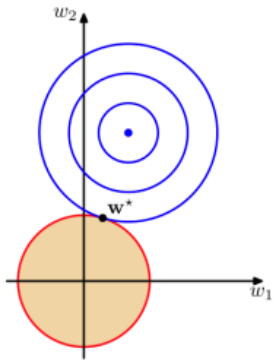
As the dimensionality of w increases, the norm ball will have increasingly more number of corners.

Summary: L2- vs L1-regularization

L2 regularization promotes **grouping** – results in equal weights for correlated features

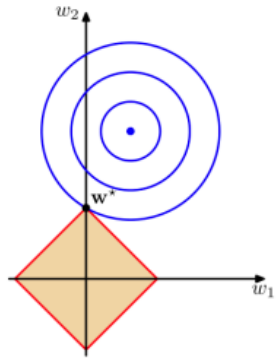
L1 regularization promotes **sparsity** – selects few informative features

L2-regularization



$$\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

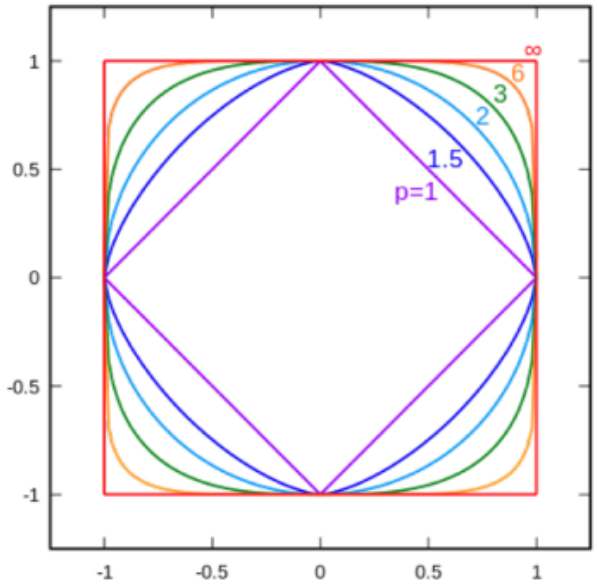
L1-regularization



$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$$

Lp-Norms

General case $\|\boldsymbol{w}\|_p$ where p can be any non-negative number



Lp norms with p < 2 promotes sparsity

Lp norms with p > 2 promotes grouping

Regularization: Expressing Preferences

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

$$x = [1, 1, 1, 1]$$
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Regularization: Expressing Preferences

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$\boxed{w_2 = [0.25, 0.25, 0.25, 0.25]}$$

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

L2 regularization prefers weights to be "spread out"

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Finding a good W

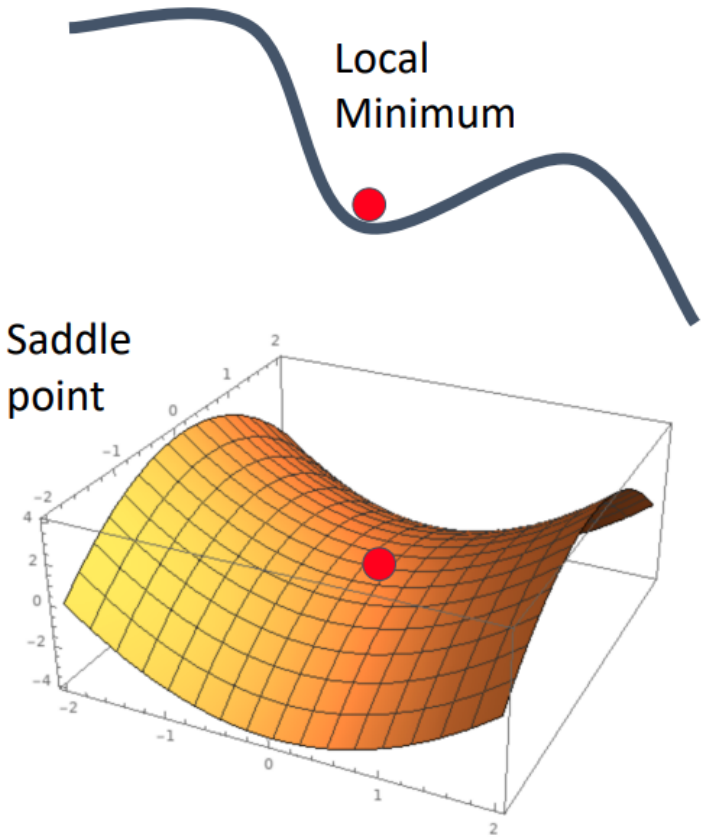
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Loss function consists of **data loss** to fit the training data and **regularization** to prevent overfitting

Problems with SGD

What if the loss function has a **local minimum** or **saddle point**?

Zero gradient, gradient descent gets stuck

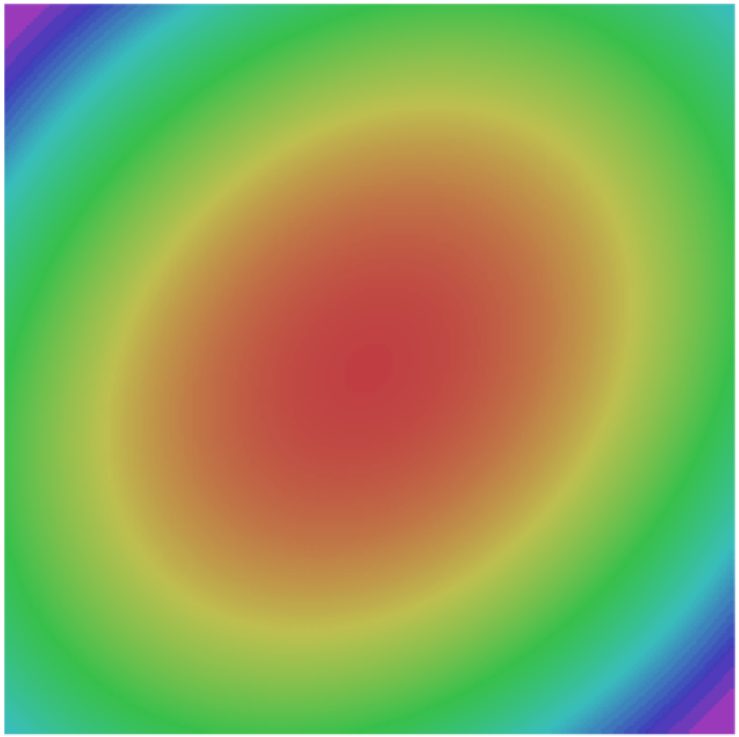


Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$



SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

SGD+Momentum

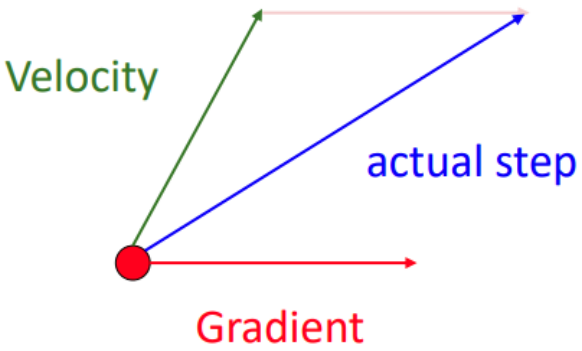
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

SGD + Momentum

Momentum update:



Combine gradient at current point with velocity to get step used to update weights

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- Build up "velocity" as a running mean of gradients
- Rho gives "friction"; typically rho=0.9 or 0.99

SGD + Momentum

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v - learning_rate * dw
  w += v
```

SGD+Momentum

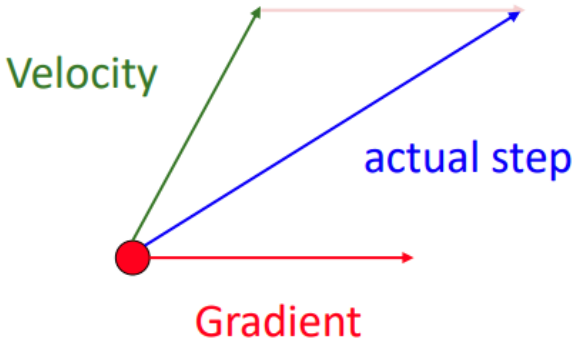
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

You may see SGD+Momentum formulated different ways, but they are equivalent - give same sequence of x

Nesterov Momentum

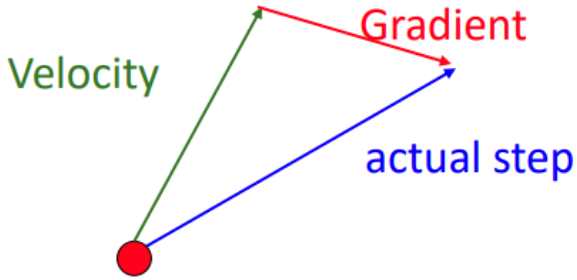
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

AdaGrad

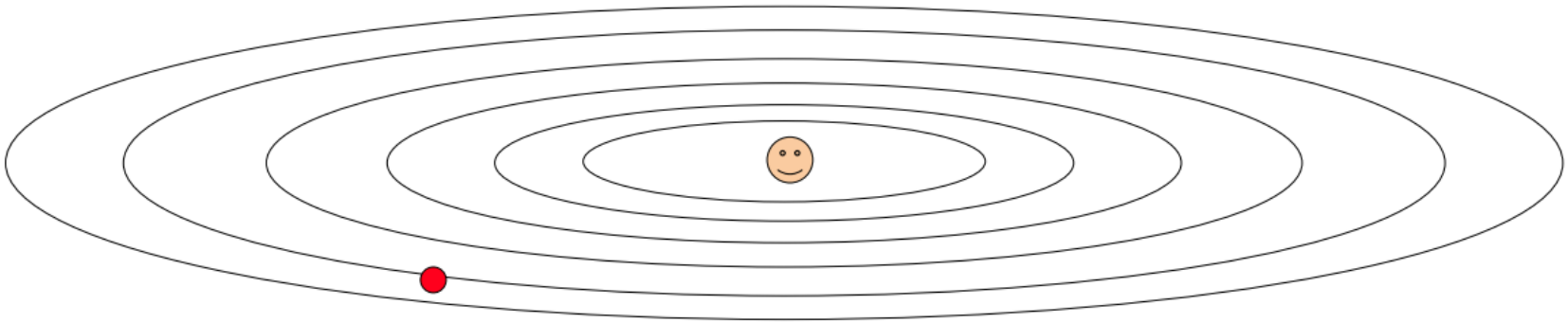
```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

"Per-parameter learning rates"
or "adaptive learning rates"

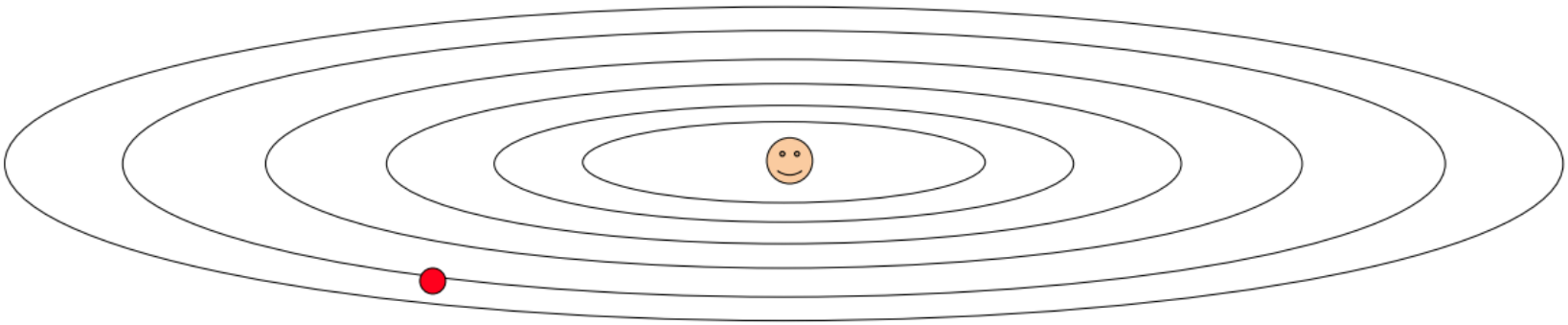
AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```



AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```



Q: What happens with AdaGrad?

Progress along “steep” directions is damped;
progress along “flat” directions is accelerated

RMSProp: "Leaky Adagrad"

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

Adam (almost): RMSProp + Momentum

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

Momentum

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

SGD+Momentum

Adam (almost): RMSProp + Momentum

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

Momentum

AdaGrad / RMSProp

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	✗	✗	✗	✗
SGD+Momentum	✓	✗	✗	✗
Nesterov	✓	✗	✗	✗
AdaGrad	✗	✓	✗	✗
RMSProp	✗	✓	✓	✗
Adam	✓	✓	✓	✓

L2 Regularization vs Weight Decay

Optimization Algorithm

$$L(w) = L_{data}(w) + L_{reg}(w)$$
$$g_t = \nabla L(w_t)$$
$$s_t = optimizer(g_t)$$
$$w_{t+1} = w_t - \alpha s_t$$

L2 Regularization and Weight Decay are equivalent for SGD, SGD+Momentum so people often use the terms interchangeably!

But they are not the same for adaptive methods (AdaGrad, RMSProp, Adam, etc)

L2 Regularization

$$L(w) = L_{data}(w) + \boldsymbol{\lambda |w|^2}$$
$$g_t = \nabla L(w_t) = \nabla L_{data}(w_t) + \boldsymbol{2\lambda w_t}$$
$$s_t = optimizer(g_t)$$
$$w_{t+1} = w_t - \alpha s_t$$

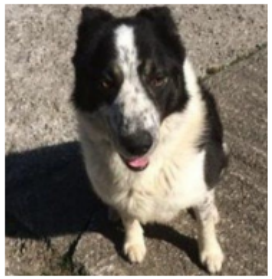
Weight Decay

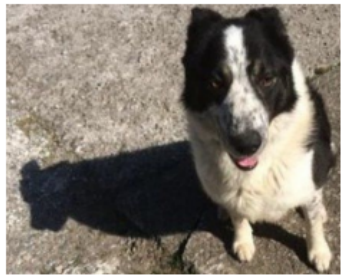
$$L(w) = L_{data}(w)$$
$$g_t = \nabla L_{data}(w_t)$$
$$s_t = optimizer(g_t) + \boldsymbol{2\lambda w_t}$$
$$w_{t+1} = w_t - \alpha s_t$$

Data augmentation

The best way to make a machine learning model generalize better is to train it with more data – however, in practice, the amount of data we have is limited.





Then, we can artificially enlarge the data by generating some variation of inputs e.g.) Flipping, cropping, translating, rotating, and etc, for image inputs

Data augmentation

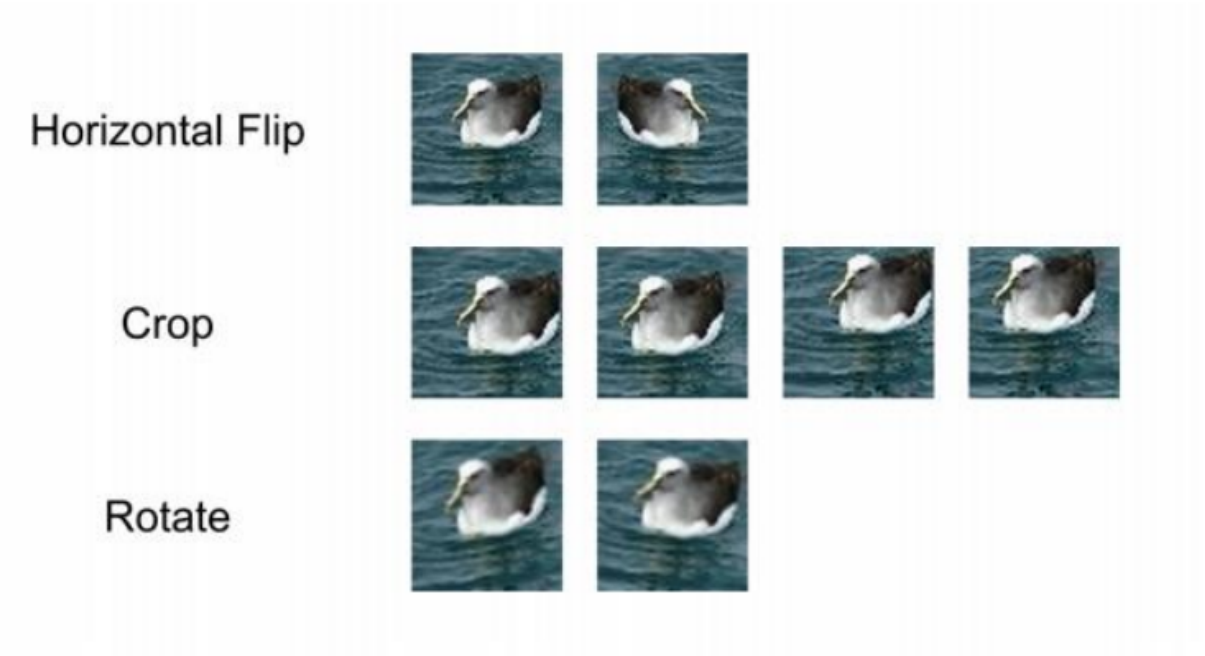


Figure from *Image Classification with Pyramid Representation and Rotated Data Augmentation on Torch 7,* by Keven Wang

Data augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied:
  - Example: classifying 'b' and 'd'
  - Example: classifying '6' and '9'

Data augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied:
  - Example: classifying 'b' and 'd'
  - Example: classifying '6' and '9'

Data augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied:
  - Example: classifying 'b' and 'd'
  - Example: classifying '6' and '9'

Data augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied:
  - Example: classifying ‘b’ and ‘d’
  - Example: classifying ‘6’ and ‘9’

Dropout

Early stopping

Next Class..